CSC 411 Quiz 2 (Makeup) — Solutions

1. If f and g are functions defined on the integers, what does it mean that a function $f = O(g)$?

   Solution. By definition, $f(n) = O(g(n))$ if there exist constants $c > 0$ and n_0 such that for every integer $n \geq n_0$,

   $$0 \leq f(n) \leq c\, g(n).$$

   Interpretation: for sufficiently large n, f grows no faster than a constant multiple of g.

2. In a graph represented using adjacency lists, both BFS and DFS maintain a collection of "discovered but not fully processed" vertices. Describe precisely how this collection differs between BFS and DFS, and explain how that difference affects the order in which vertices are explored.

   Solution. Both algorithms maintain a frontier, but with different data structures:

   - **BFS**: frontier is a **queue** (FIFO). The earliest discovered pending vertex is processed next.
   - **DFS**: frontier is a **stack** (LIFO), explicit or recursive call stack. The most recently discovered pending vertex is processed next.

   Consequences:

   - BFS explores vertices in nondecreasing distance (level by level) from the start.
   - DFS explores one path as deeply as possible before backtracking.

3. Suppose you run BFS and DFS on the same connected graph starting from the same source vertex. Which algorithm is guaranteed to find a shortest path (measured in number of edges) to a target vertex? Explain why this guarantee holds for one algorithm but not the other.

   Solution. BFS is guaranteed to find a shortest path in an unweighted graph (where edge cost is 1 per edge).

   Why: BFS visits vertices in layers:

   $$L_0 = \{s\}, \quad L_1, \quad L_2, \ldots$$

   where L_k is exactly the set of vertices at distance k from s. The first time BFS discovers a vertex v, it must be via a path with the minimum number of edges.

   DFS has no such guarantee because it may follow a long branch first and reach the target via a non-shortest path before it ever explores shorter alternatives.

4. Suppose we run BFS on an undirected graph but remove the visited array entirely. Instead, whenever we dequeue a vertex, we simply enqueue all of its neighbors without checking whether they were seen before. Will the algorithm terminate on all finite graphs?

   Solution. No, it will **not** terminate on all finite graphs.

   Counterexample: graph with two vertices u and v and one undirected edge $\{u, v\}$, start at u.

- Dequeue u, enqueue v.
- Dequeue v, enqueue u.
- Dequeue u, enqueue v.
- Repeat forever.

Without a visited check, cycles cause endless re-enqueueing, so the queue never empties (except in special acyclic/isolated cases).